Crucial Components in Probabilistic Inference Pipelines

Dimitar Shterionov
Computer Science, KULeuven
Celestijnenlaan 200A, bus 2402
Heverlee, Belgium
dimitar.shterionov@cs.kuleuven.be

Gerda Janssens
Computer Science, KULeuven
Celestijnenlaan 200A, bus 2402
Heverlee, Belgium
gerda.janssens@cs.kuleuven.be

1. BACKGROUND

1.1 ProbLog

ProbLog [7, 10] is a general purpose Probabilistic Logic Programming (PLP) language. It extends Prolog with uncertain knowledge encoded as probabilistic facts. A probabilistic fact, $p :: f$ states that the fact f is true with probability p. The following is a ProbLog program encoding a probabilistic graph:

```
0.6::e(a,b).  0.3::e(a,c).  0.8::e(b,c).  0.4::e(b,d).  0.7::e(c,d).
p(X, Y):- e(X, Y).
p(X, Y):- e(X, X1), p(X1, Y).
```

The fact $p_i :: e(a_i, b_i)$. encodes the edge between nodes a_i and b_i which exists with probability p_i. The `p/2` predicate defines the ("path") relation between two nodes: a path exists, if two nodes are connected by an edge or via a path to an intermediate node.

Each probabilistic fact can be either true or false in different models of the ProbLog program. A model of a ProbLog program, called a *possible world* contains all ground atoms of the initial ProbLog program with a specific truth value assignment. ProbLog defines a distribution over the possible worlds as follows: the probability of a possible world is the product of the probabilities of all probabilistic facts which are true in the possible worlds and (1-the probability) of facts which are false in the world.

A query atom q is true in a subset of the possible worlds. The *marginal* probability of q is the sum of the probabilities of all worlds in which q is true.

1.2 Weighted Model Counting by Knowledge Compilation

Exhaustively enumerating all possible worlds in order to perform inference is almost always impossible. ProbLog uses knowledge compilation to reduce the inference task to an efficient Weighted Model Counting (WMC). The approach can be summarized as a two-step process in which a ProbLog program L together with a set of query atoms Q is (i) converted into a weighted Boolean formula on which (ii) weighted model counting is performed efficiently.

Model Counting is the process of determining the number of models (the Model Count) of a formula φ. The *Weighted* Model Count (WMC) of a formula φ is the sum of the weights associated with each model of φ. This coincides with the semantics of ProbLog when a ProbLog program L is equivalent to a Boolean formula φ. Efficient algorithms [5] for WMC have found their place in ProbLog.

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

2. INFERENCE PIPELINE

ProbLog uses a **pipeline** of transformation subprocesses in order to reduce the inference task to a weighted model counting problem: first a propositional instance (also referred to as the *grounding*) is generated by grounding L. This step ignores the probabilistic information of a ProbLog program, i.e. the probability label of each probabilistic fact. The grounding of L is then converted to an equivalent (with respect to the models) Boolean formula. Next, the Boolean formula is compiled into a *negation normal form* (NNF) with certain properties which allow efficient model counting. This NNF is then converted to an arithmetic circuit which is associated with the probabilities of L and weighted model counting is performed.

Initially, in ProbLog1 [7], by using knowledge compilation to *Reduced Ordered Binary Decision Diagrams* (ROBDDs) [1] probabilistic inference was reduced to a tractable problem. Later, [8] illustrates another approach for ProbLog inference where a grounding of the initial program is converted into a propositional formula in conjunctive normal form (CNF) which is then compiled into a *smooth, deterministic, Decomposable Negation Normal Form* (sd-DNNF) [6]. Here we present and analyze the various tools or algorithms, i.e. components that can be chained together in order to compile a ProbLog program (together with a set of queries) into an sd-DNNF or a ROBDD and compute the WMC. Figure 1 gives an overview of different subprocesses and how they can be linked in order to form an inference pipeline. Up to our knowledge such a complete analysis of the ProbLog inference pipelines is not yet presented in earlier works.

2.1 Grounding

In order to avoid the complete grounding of a program ProbLog focuses on the part which is relevant to an atom of interest. A ground ProbLog program is relevant to an atom q if it contains only relevant atoms and rules. An atom is relevant if it appears in some *proof* of q. A ground rule is relevant with respect to q if its head is a relevant atom and

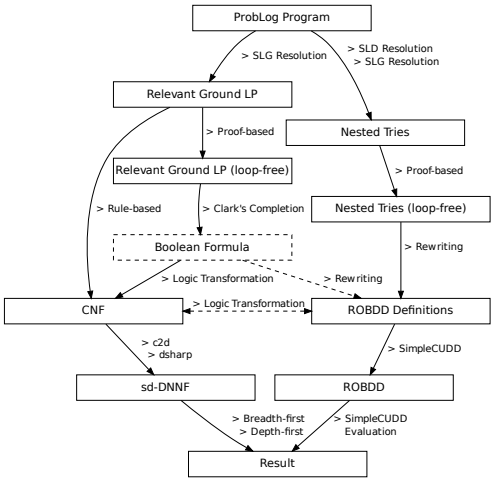

Figure 1: ProbLog pipelines. Directed edges define the processes which take place for each inference step. Solid edges define a ProbLog1 or ProbLog2 pipeline. Dashed edges state a transformation which allows to switch from a ProbLog1 to a ProbLog2 pipeline or vice-versa. Nodes specify input/output. Dashed nodes indicate complementary data formats. The input ProbLog program may contain queries.

its body consists of relevant atoms. That is, the relevant ground program captures the distribution $P(q)$ entirely [8].

To determine the relevant grounding a natural mechanism is SLD resolution. SLD resolution generates a tree of ground atoms which unify with a (sub)goal. Collecting all successful branches of that tree determines the relevant ground program. A proof is a conjunction of literals derived by traversing the SLD tree starting from the root, i.e., the query, and ending at an empty clause. Naturally, all proofs to a query form a disjunction and therefore, can be represented as a Boolean formula in DNF. In case of cyclic programs the SLD tree becomes infinite. In order to avoid complications caused by cycles SLG resolution [2] (that is, SLD with tabling) can be used instead generating a forest of (nested) tries.

We distinguish between two representations of the relevant grounding of a ProbLog program. ProbLog1 uses the **nested trie structure** as an intermediate representation of the collected proofs. If SLD resolution is used (that is, no tabling is invoked)[1] there is only one trie which corresponds to the SLD tree. ProbLog2 considers the **relevant ground logic program** with respect to a set of (query) atoms.

2.2 Boolean Function of the Grounding

Logic Programs (LP) use the Closed World Assumption (CWA), which basically states that if an atom cannot be proven to be true, it is false. In contrast, First-Order logic (FOL) has different semantics. Consider the (FOL) theory $\{q \leftarrow p\}$ which has three models: $\{\neg q, \neg p\}$, $\{q, \neg p\}$ and $\{q, p\}$. Its syntacticly equivalent LP (`q :- p.`) has only one model, namely $\{\neg q, \neg p\}$. In order to generate a Boolean Function from nested tries or a relevant ground LP it is required to make the transition from LP semantics to FOL semantics. When the grounding does not contain cycles it suffices to take the Clark's completion of that program [9]. When the grounding contains cycles it is proven that the Clark's completion does not result in an equivalent Boolean function [9]. To handle cyclic groundings ProbLog employs one of two methods. The first one (the **proof-based** approach) [11] basically removes proofs containing cycles as they do not contribute to the probability. The second one (the **rule-based** approach) is inherited from the field of Answer Set Programming. It rewrites a rule with cycles to an equivalent rule and introduces additional variables in order to disallow cycles [9].

Once the cycles are handled, ProbLog1 generates **ROBDD definitions**. A ROBDD definition [11] is a formula with a head and a body, linked with equivalence. The body of a ROBDD definition contains literals and/or heads of other ROBDD definitions combined by conjunctions or disjunctions. The logic operators are translated to arithmetic functions. In the case of ProbLog2 the relevant ground LP is converted to a formula in **CNF**. The ground LP can also be rewritten to **ROBDD definitions** and vice versa.

2.3 Knowledge Compilation and Evaluation

Knowledge compilation is the process in which a Boolean function is compiled to a negation normal form (NNF) with certain properties [6] to ensure correct (with respect to ProbLog inference) weighted model counting. In ProbLog's inference pipelines two target compilation languages have been exploited so far: (i) **ROBDDs** [1] common for ProbLog1 and (ii) **sd-DNNFs** [6] employed by ProbLog2.

To compile a Boolean function as a ROBDD ProbLog implementations use **SimpleCUDD** (`www.cs.kuleuven.be/~theo/tools/simplecudd.html`). Compiling to sd-DNNF is done with the **c2d** [3, 4] or **dsharp** [12] compilers.

In the evaluation step, the compiled Boolean circuit is traversed in order to compute the probabilities (i.e. the WMC) for the given query(ies). ProbLog employs two approaches to traverse sd-DNNFs: **Breadth-First** and **Depth-First**; and a **Depth-First** approach to traverse ROBDDs.

3. CRUCIAL COMPONENTS

Figure 1 gives an overview of the interchangeable components which constitute possible ProbLog pipelines. The link between different components depends on the compatibility of the output of a preceding subprocess with the input requirements of the next one. For example, c2d cannot compile ROBDD definitions as it requires CNFs. We now compare the different components for each step in the inference pipeline in order to determine which of them are crucial for the overall system's performance and usability.

The two grounding components are based on the same mechanism – SLG (or SLD) resolution. Due to the efficiency of the SLG (or SLD) resolution the grounding has little impact on the overall performance. Nested tries encode a disjunction of the collected proofs, i.e. a formula in DNF. As such they are more suitable for pipelines which use compilation to ROBDDs. The ground LP is preferable for converting into a CNF and then compilation to sd-DNNFs.

The conversion to a Boolean function is of high importance for the overall performance. The reason is that the next step (the knowledge compilation) is computationally the hardest and its performance strongly depends on the input Boolean function. We observe two phenomena related to

[1]ProbLog1 allows the user to select whether to use tabling or not. ProbLog2 always uses tabling.

the Boolean function which are crucial for the performance of a ProbLog pipeline.

Observation 1. The rule-based conversion can perform better than the proof-based [9]. The output CNF though is more complex. Compiling such CNFs to sd-DNNFs and subsequently evaluating the sd-DNNs is cumbersome and can lead to extremely slow performance.

Observation 2. Rewriting a Boolean function in CNF into ROBDD definitions is inefficient.

Example *1. Consider the Boolean formula:*

$$(a \iff (b \wedge c)) \wedge (b \iff (p \vee q)) \wedge (c \iff \neg r).$$

Following are its equivalent representations as a CNF and ROBDD definitions:

CNF:	*ROBDD definitions:*
$(a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \wedge (\neg a \vee c) \wedge$	`b = p + q`
$(\neg b \vee p \vee q) \wedge (b \vee \neg p) \wedge (b \vee \neg q) \wedge$	`c = ~r`
$(\neg c \vee \neg r) \wedge (c \vee r)$	`a = b * c`

Example *2. A CNF formula can be translated into ROBDD definitions and vice-versa. The following ROBDD definitions are generated from the CNF in Example 1 and are equivalent to the Boolean formula in Example 1:*

ROBDD definitions 2:

```
L1 = ~b + p + q   L2 = b + ~p       L3 = b + ~q   L4 = c + r
L5 = ~c + ~r      L6 = a + ~b + ~c  L7 = ~a + c   L8 = ~a + b
L9 = L1 * L2 * L3 * L4 * L5 * L6 * L7 * L8
```

The Clark's completion of a cycle-free logic program is a formula similar to the one in Example 1. This formula can easily be converted to CNF as well as to ROBDD definitions. Example 1 shows that a CNF representation of such a formula is less succinct ([6]) than the representation as ROBDD definitions. If though a CNF formula is converted to ROBDD definitions as in Example 2 the ROBDD script blows up in size. For the overall performance of a pipeline it is crucial to avoid components which perform such a transformation. This phenomenon is discussed among others in [13]. In [8] the authors consider a ProbLog pipeline in which a CNF formula is transformed into ROBDD definitions as shown in Example 2, i.e. a relevant ground LP is first converted to a Boolean circuit in CNF which subsequently is converted to a ROBDD script. Their experiments confirm that such an approach is inefficient for ProbLog inference.

We determine that the conversion to Boolean function is the component with highest impact on the inference pipeline.

Regarding the compilation we ought to note the differences between ROBDDs and sd-DNNFs which make the one preferable to the other. [6] states that sd-DNNFs are at least as succinct as ROBDDs. ROBDDs, though, posses some properties which make them preferable to sd-DNNFs in a ProbLog pipeline. They allow polytime Boolean transformations, i.e. bounded conjunction, bounded disjunction and negation [6]. Therefore, compiling to ROBDDs can be performed in an efficient bottom-up manner. The size of an ROBDD strongly depends on the order variables are processed. Dynamic variable reordering allows the transformation of ROBDDs during the compilation stage when new variables are presented. The bottom-up compilation, the dynamic reordering and the succinct representation of the Boolean formula as ROBDD definitions (see Observation 2) are the main factors for ProbLog pipelines with ROBDDs to perform faster than with sd-DNNFs.

In this work we analyzed different components of a ProbLog inference pipeline. We determined that the Boolean function conversion has a crucial impact on the performance of the inference pipeline. Our future goals revolve around optimizing the Boolean function so that the cost for knowledge compilation can be reduced. Furthermore, we need to empirically support our analysis by performing extensive tests on the different pipelines.

References

[1] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[2] Weidong Chen, Terrance Swift, and David Scott Warren. Efficient top-down computation of queries under the well-founded semantics. *J. Log. Program.*, 24(3):161–199, 1995.

[3] Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In Rina Dechter and Richard S. Sutton, editors, *AAAI/IAAI*, pages 627–634. AAAI Press/MIT Press, 2002.

[4] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 328–332, 2004.

[5] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. Chapter 12.

[6] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[7] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: a probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[8] Daan Fierens, Guy Van Den Broek, Joris Renkens, Dimitar Shterionov, Bern Gutmann, Ingo Thon, Gerda Janssens, and Luc de Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming, Special Issue on Probability, Logic and Learning*, 2013.

[9] Tomi Janhunen. Representing normal programs with clauses. In *In Proc. of the 16th European Conference on Artificial Intelligence*, pages 358–362. IOS Press, 2004.

[10] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vítor Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11:235–262, 2011.

[11] Theofrastos Mantadelis and Gerda Janssens. Dedicated tabling for a probabilistic setting. In Manuel V. Hermenegildo and Torsten Schaub, editors, *ICLP (Technical Communications)*, volume 7 of *LIPIcs*, pages 124–133. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[12] Christian Muise, Sheila A. Mcilraith, J. Christopher Beck, and Eric Hsu. DSHARP: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*, 2012.

[13] Antoine Rauzy, Eric Châtelet, Yves Dutuit, and Christophe Bérenguer. A practical comparison of methods to assess sum-of-products. *Rel. Eng. & Sys. Safety*, 79(1):33–42, 2003.